Talking about how to prototype an embedded system is one thing but actually doing it is a totally different story. Over the course of the coming months, I’m going to be putting together a series of hands-on articles demonstrating how to prototype and create a production-intent industrial controller that connects to the internet, i.e., an IoT industrial controller. You’re invited to join me on the journey, beginning with exploring prototyping approaches.

Building embedded systems is not rocket science, but neither is it usually a nice stroll through the park. Because electronic systems and software are complicated (and with each passing day the complexity is only increasing) developers need methodologies for identifying high risk system features and for rapidly gaining insights into the system in order to properly manage the risk. Prototyping an embedded system is a great way to reduce system risk through experimentation and sometimes even plain old trial and error.

Rapid prototyping has other benefits, as well. Humans are optimistic and we often expect success even under the direst of circumstances. But if a project is to fail, as sometimes they do, we want them to fail as quickly as possible. Rapid prototyping can be used as a tool to verify market conditions and test technologies under development. A development kit, some jumpers, and a quick code hack can help quickly verify a flippant time and cost estimate made at the outset of the project so that a real estimate can be provided or the project can be scrapped before nearly putting a company out of business.

The challenge of embedded system design doesn’t end at proving that an idea or time estimate is accurate, though. The challenge is sometimes convincing the manager or business owner that the working proof-of-concept is just that – a concept. The system works, but only under carefully controlled conditions. So in the coming months we’ll not only explore techniques on how we can rapidly prototype our embedded system, we’ll look at how to convert the prototype to a production-intent system as rapidly as possible.

The first step on this journey is to choose your path. There are many different avenues that a developer can follow in prototyping an embedded system. One great place to start is to use low cost development kits or even embedded system platforms. A platform will usually provide low level driver code, middleware, and sometimes even example code snippets. There are a number of embedded platforms available today, such as the Renesas Synergy Platform, but probably the best well known platforms for rapid prototyping are Arduinos.

Arduino development kits are inexpensive, costing as little as $25 to as much as $70 depending on the onboard processing capability. The power of Arduino is partially in its open source software, which provides common routines and functions no matter which board a developer is using. The software is legible enough that even hobbyists, artists, and electrical engineers can write code quickly and easily. Arduino's high rate of acceptance and standard hardware interface has also
allowed a wide variety of shields and add-on boards to be created for nearly any application imaginable. We’ll explore Arduinos in greater detail in a future article.

Along with a range of platform hardware choices, developers have a range of programming languages they could work with. For the longest time the programming language for embedded systems had by default been C. But while the C programming language isn’t going anywhere anytime soon, as new platforms are released and example or production code becomes available in other languages such as C++ (mbed for example) or even Python, developers may discover the number of projects developed in C will begin to whither.

Exploring how to run Python on a microcontroller could be an interesting exercise for an industrial controller developer. Using a high level language like Python, that is easy to learn, could be used by anyone looking to simply customize the hardware controller for a specific application with little to no knowledge of the underlying hardware. We’ll explore languages and the use of Python in greater detail in the series as well.

As our journey begins, I have an endless list of ideas and directions that this series can take. In order to maximize the benefit to you the reader, I could use some guidance along the path. If you can provide comments about what you would like to hear and learn about, it will be most helpful. To that end, let me ask this: What concepts or technologies would you like to see built into the IoT industrial controller I’ll develop through this series? Post your ideas and suggestions in the comment field below!

*Jacob Beningo is principal consultant at Beningo Engineering, an embedded software consulting company. Jacob has experience developing, reviewing and critiquing drivers, frameworks and application code for companies requiring robust and scalable firmware. Jacob is actively involved in improving the general understanding of embedded software development through workshops, webinars and blogging. Feel free to contact him at jacob@beningo.com, at his website www.beningo.com, and sign-up for his monthly Embedded Bytes Newsletter [here](#).*

**Also see:**

- 7 tips for creating a reliable embedded system
- Prototyping methods
- IoT dev kit enables rapid prototyping
- Why, When and How: The basics of embedded systems prototyping
- Embedded systems face design, power, security challenges